On Knowledge Transfer from Cost-based Optimization of Data-centric Workflows to Business Process Redesign

Georgia Kougka[1], Konstantinos Varvoutas[1], Anastasios Gounaris[1], George Tsakalidis[2], and Kostas Vergidis[2]

[1] Department of Informatics,
Aristotle University of Thessaloniki, Greece
{georkoug,kmvarvou,gounaria}@csd.auth.gr
[2] Department of Applied Informatics,
University of Macedonia, Thessaloniki, Greece
{giorgos.tsakalidis,kvergidis}@uom.edu.gr

Abstract. This work deals with redesigning business process models, e.g., in BPMN, based on cost-based optimization techniques that were initially proposed for data analytics workflows. More specifically, it discusses execution cost and cycle time improvements through treating business processes in the same way as data-centric workflows. The presented solutions are cost-based, i.e., they employ quantitative metadata and cost models. The advantage of this approach is that business processes can benefit from recent advances in data-intensive workflow optimization similarly to the manner they nowadays benefit from additional data analytics areas, e.g., in the area of process mining. Concrete use cases are presented that are capable of demonstrating that even in small, more conservative cases, the benefits are significant. The contribution of this work is to show how to automatically optimize the model structure of a given process in terms of the ordering of tasks and how to perform resource allocation under contradicting objectives. Finally, the work identifies open issues in developing end-to-end business process redesign solutions with regards to the case studies considered.

1 Introduction

Modern *Business Processes (BPs)* constitute a key part of businesses and their modeling, execution and evolution are critical aspects in *Business Process Management (BPM)*. The BPM is defined as a body of methods, techniques and tools to discover, analyze, redesign, execute and monitor these business processes [17]. In such a context, a fundamental role of BPM is to adapt business functions to the requirements of each business's customer, allocate efficiently the business resources, target at both improving the quality of services delivered while keeping the cost low, keep the processes as simple and flexible as possible, and so on. In order to fulfill these demanding and often contradicting business requirements, the behavior of business processes is closely monitored after their deployment, so as to gain insights that were unknown at design time. Such information can be used for adapting the process's design to improve if not optimize its efficiency.

The optimization of the BPs, as considered in this work, refers to (incremental) BP *redesign*, which is an essential part of BPM lifecycle and a necessary action to ensure the best schedule, coordination and execution of business activities [23]. We assume that it is trivial to design an initial BP model, which is then frequently refined to optimize certain metrics and/or adapt to changing conditions. The re-ordering of the BPs's tasks and re-allocation of the available resources are key examples of BP optimization actions. Nevertheless, up to date, these optimization actions are applied mostly manually based on the experience of the business users and the help of BPMN software tools [3, 4, 2, 1], which do not support fully-automated optimization solutions and require human interaction. Consequently, there is a demanding need to introduce automated optimization solutions, where automation covers not only the monitoring of the process and the identification of an issue but also the modification of the process model structure, such as changing the order in which tasks are placed in the model and executed. The aim of this work is to introduce techniques for automated BP optimization inspired by a set of well-investigated algorithms proposed for data-centric workflow optimization [30, 27, 26]. This is aligned to several recent attempts to further blend advanced data analytics with business processes [44]. More specifically, in this work, it is demonstrated how to automatically optimize the task ordering, thus modifying the model structure of a given process, and how to perform resource allocation under contradicting objectives.

BP redesign is a well investigated area; nevertheless, it is mostly based on heuristics (see Appendix in [17] for a complete list) with a lack of automated optimization solutions [52] that can be directly run on initial model structures. BP redesign covers both evolutionary and revolutionary approaches (see Ch. 8 in [17]); our work fits better the evolutionary paradigm, where we restructure an existing process model or we perform resource allocation in an informed manner. Cost-based optimization solutions for BPs have been explored but they are incomplete from the algorithmic point of view and inferior to those in the area of optimization of data-centric workflows [42, 29]. An example that we particularly deal with in this work concerns ordering knock-out tasks, i.e., tasks that may lead to immediate process termination; in this context, to minimize total cost, the underpinnings of the works in [5] (regarding BPs) and [26] (regarding workflows for data analytics) are the same; however, the latter goes one step beyond and provides a more complete algorithmic solution in that it takes into account a broader range of workflows containing additional tasks apart from knock-out ones and arbitrary dependency constraints between them, as is the norm.

Before delving into details, it is important to distinguish between the meaning of the term "data flow" in the different communities. In the broader data-management community, data flows denote data-centric workflows, e.g., as supported by modern tools, such as Apache Spark [55]. However, in business processes, data flow is used as complementary to control flow: the former emphasizes on the artifacts, such as documents, required to perform a task, whereas the latter indicates when tasks and events should occur [17]. Our work advocates leveraging advances in data-centric workflows to optimize the control flow of BPs; optimizing the data flow in BPs. e.g., as in [25], is a complementary area to our proposal.

The optimization of BPs may consider multiple critical objectives, such as the minimization of the economic and time cost ensuring that the quality of service is high. Other examples of optimization objectives are the response time and the resource utilization. Similarly, the means to improve on these metrics vary: from dropping unnecessary tasks to reordering tasks and modifying the resource allocation. In this work, we will focus on automated BP optimization considering multi-criteria improvement, such as the minimization of execution cost and cycle time of BPs using a principled approach to investigating alternative task orderings and task allocation solutions. The contribution of this work is to present complete solutions and anticipated benefits to the problem of reordering knock-out tasks and modifying the allocated resources to business processes, and finally, understanding better the issues involved in coupling BP redesign and data-centric workflow optimization more tightly.

The structure of the manuscript is described as follows. We present the related work of optimizing BPMN processes in Section 2. We describe motivational examples in order to highlight the need for adopting cost-based optimization solutions in Section 3. The metadata definition and the necessary background for applying the optimization algorithms, along with the optimization metrics are presented in Section 4. The next section shows how knock-out processes can be treated in a principled manner and provides insights into the expected benefits in a real scenario; in addition, we provide an approach to bi-objective optimization using another real scenario. Software implementation issues are treated in Section 6 along with a detailed discussion of open issues, whereas the following section concludes this work.

2 Related Work

Several optimization techniques to support BP redesign have been proposed. For example, the authors in [17] summarize all the heuristics in order to refine an existing process or eliminate redundant tasks by merging or removing processes (without presenting concrete algorithms). Similarly, good practices, such as using as few elements in the model as possible and employing models as structured as possible avoiding OR routing elements appear in [34]. Similarly, the work in [18] advocates following high-level business process improvement patterns. Another redesign approach is to exploit queue theory to emulate human or machines that execute tasks [15] with a view to driving decisions regarding task re-ordering, resource allocation and task implementation. Contrary to the use cases we consider, the focus of the work in [15] is on how to merge different tasks into groups that can be handled by the same team of actors. Additionally, there are proposals considering variant optimization objectives, such as the techniques in [5], where a set of heuristics is introduced for optimizing the metrics of resource utilization, maximal throughput and execution (cycle) time. These heuristics consider changing the relative ordering of tasks, enforcing parallel execution and task merging; essentially, they constitute building blocks for developing automated algorithms rather than proposing full algorithms. Finally, another interesting approach to redesigning BPs is the re-configuration of the BP models that comprise several variants, e.g. so that different tokens in the same BPMN flow may follow different paths as proposed in [32].

Regarding data-centric workflows, a lot of effort is towards finding the best sequential order of flow tasks [27, 26] to minimize the sum of the costs of these tasks. Another important optimization mechanism examined in the state-of-the-art is the selection of the best execution option of a process, choosing between multiple human and/or physical resources/alternatives [30]. Additionally, there are proposals that optimize different single objectives than the sum of the costs, such as the improvement of the sum of the task costs of the critical flow execution path [8] or the task ordering to maximize the utilization of each execution processor [16]. All these proposals aim to optimize a single criterion, but there are also proposals that target multi-objective data flow optimization, such as the algorithms in [46, 47] that consider the sum of the task costs along with the reliability, in the form of fault tolerance.

In addition, there are several recent proposals on data analytics-driven solutions for BPM problems. Examples include process mining and discovery, e.g., [7, 10, 9, 56], and querying-based [40] process enhancements. Process mining in particular is both orthogonal and complementary to our research. More specifically, process mining may be used to extract the necessary statistical metadata, e.g., [6](Ch. 8.4) and precedence constraints, e.g., [48, 41] to be presented in the next sections, but this direction is not covered in this work.

A significant part of recent research in BPs targets variability between process models aiming at the same high-level objectives [45, 54]. For example, the work in [45] is motivated from the fact that the same goal in different municipalities is performed using different equivalent processes and, to manage such variability, it introduces the configurable process trees. This methodology allows a specific set of process models to be selected according to several criteria. The main difference with our work is that we explore process model alternatives that are not known a-priori where the search space may be exponential, e.g. when examining the ordering of knock-out tasks for which any ordering is valid. In such a context, proposals like [14] deal with the problem of extracting alternative models, whereas the issue of assessing the quality of different process model configurations [13] and quantifying the differences between process models [11] have also been explored. Finally, our work relates to declarative process models [38, 37, 33]; e.g., our task ordering solution can be seen as a promising means to derive executable model structures out of such declarative models.

3 Targeted Use Cases

This section presents the use cases we primarily target.

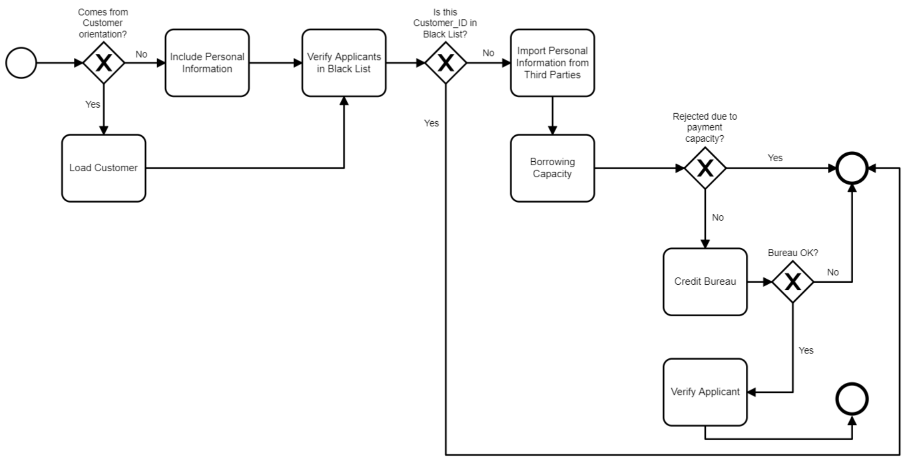

Comes from Customer orientation?
No
Yes
Include Personal Information
Load Customer
Verify Applicants in Black List
Is this Customer_ID in Black List?
No
Yes
Import Personal Information from Third Parties
Borrowing Capacity
Rejected due to payment capacity?
Yes
No
Credit Bureau
Bureau OK?
No
Yes
Verify Applicant

Fig. 1. A model representing a loan verification procedure.

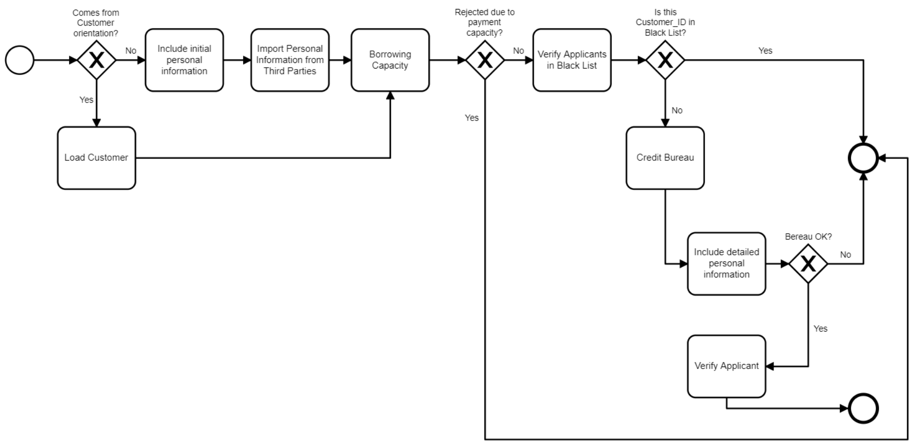

Comes from Customer orientation?
No
Yes
Include initial personal information
Import Personal Information from Third Parties
Borrowing Capacity
Load Customer
Rejected due to payment capacity?
No
Yes
Verify Applicants in Black List
Is this Customer_ID in Black List?
Yes
No
Credit Bureau
Include detailed personal information
Bereau OK?
No
Yes
Verify Applicant

Fig. 2. Task decomposition and re-ordering examples in a business process in BPMN in Figure 1.

3.1 Loan Application

In Figure 1, we show a BP example of a typical bank loan application processing. This BP consists of seven tasks, namely and two start/end events. These tasks include three checks (*Verify Applicants in Black List, Borrowing Capacity, Credit Bureau*) and a final verification phase. The checks, according to the outcome of which the loan application proceeds or is rejected, are typical examples of knock-out tasks in the sense that if the application does not pass them, the process terminates.

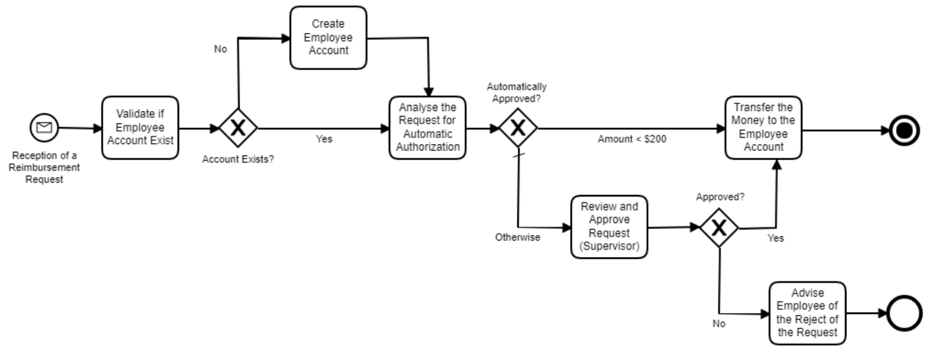
Task preceding	**Task succeeding**
Include Initial Personal Information or Load Customer	Verify Applicants in Black List, Import Personal Info from Third Parties, Borrowing Capacity, Credit Bureau, Verify Applicant
Import Personal Information from Third Parties	Borrowing Capacity, Credit Bureau
Verify Applicants in Black List, Borrowing Capacity, Credit Bureau	Verify Application

Table 1. The dependency constraints of the BPMN scenario of Figure 2.

Assume that the second knock-out task, i.e., the one referring to the borrowing capacity, is much more selective than the first one, i.e., the one referring to the black list; in such a case, it is beneficial to swap the first two checks. The interesting part is that, in order to run the second check, another task, namely *Import Personal Information from Third Parties*, is required to be moved upstream as well. Let us further assume that in this scenario, the checks are automated and each one takes less than 1 minute to complete, whereas an employee requires 25 minutes per customer in order to fill all the necessary forms with the personal details of a customer (*Include Personal Information*). This implies that the complete BP execution delays, until this task is completed while, in case of application rejection, not all the info is needed. Therefore, one can reduce the cycle time through (i) decomposing the complex task of *Include Personal Information* to two simpler tasks, only the first of which is required to run the subsequent three checks; (ii) moving the non-necessary part of the detailed application information either after all knock-out tasks or in parallel with them; an example is shown in Figure 2. Such re-ordering and/or decomposition optimization actions can yield improvements regarding both the resource consumption and the process cycle time.

A main question that arises is as to whether we can be sure that the semantics of the process have not been modified and re-ordering tasks does not affect the rationale of the process. To ensure these requirements, we need to respect the precedence (or dependency) constraints between tasks. These constraints for the running example are shown in Table 1, and they should be interpreted as follows: in any path from a source event to an end event that contains tasks in the same row of the table, the tasks in the left column should be placed beforehand the ones in the right column. In other words, in each row, the tasks in the right cell depend on the output of the tasks in the left cell. Therefore, we use the terms dependency and precedence constraints interchangeably in this work.

The contents of Table 1 can be represented as a graph with an edge starting from a task in the left cell pointing to a task in the right cell in the same row. The full list of the precedence constraints is derived from the transitive closure in such a graph. These constraints can be derived manually with the help of process analysts and domain experts; automated extraction, even in data-centric



Reception of a Reimbursement Request
Validate if Employee Account Exist
No
Create Employee Account
Account Exists?
Yes
Analyse the Request for Automatic Authorization
Automatically Approved?
Amount < $200
Transfer the Money to the Employee Account
Otherwise
Review and Approve Request (Supervisor)
Approved?
Yes
No
Advise Employee of the Reject of the Request

Fig. 3. Another business process model example that is amenable to optimizations.

workflows, is still not a mature area [42].[3] In this case study, we assume that the process designer also provides the constraints; issues related to constraints are further discussed in Section 6.

3.2 Reimbursement Request

An additional motivational example is presented in Figure 3, where we show a BP example of an *Employee Expense Reimbursement Request* process[4]. This BP consists of six tasks that are required to validate, analyse, approve and pay an expense statement submitted by an employee of a business. Despite its simplicity and the fact that there are no knock-out tasks, this process is also amenable to optimizations regarding the cycle time. For example, if the bottleneck is the review from a supervisor for amounts larger than $200, then more resources can be allocated to this task leading to a trade-off regarding cycle time and resource consumption (and monetary cost). In addition, we can move the supervisor review in parallel with the account existence check to save time. The goal is to employ automated solutions that explore all such alternatives.

4 Definitions and Optimization Objectives

We consider a BP to be a sextuple $P = (V, G, E, A, I, O)$, where V is the non-empty set of tasks, G is the set of gateways, E the set of edges connecting the tasks and the gateways, A is the set of actors executing the tasks, I is the set of input events, and O is the output events.

[3] Note that these constraints refer to the process model structure; additional execution constraints, e.g., two tasks share the same resource and thus cannot run simultaneously, affect the cost models that quantify the optimization objectives (see also Section 4.2).

[4] taken from https://www.businessprocessincubator.com/

We restrict ourselves to a subset of BPMN elements, comprising simple, loop, multiple instance, compensation and adhoc tasks, (event) subprocesses, AND/OR/XOR-splits, and start, end and wait timer events (wait timer events are considered as an annotation to edges). For all these elements, a mapping to a DAG (directed acyclic graph) is possible [19], even when cycles and loops exist in the model. Working with a subset of BPMN elements is common, e.g., as in [43, 22], to reduce the complexity of the proposed techniques. We consider both well-structured processes [24], e.g., derived through the process in [39] and unstructured ones. Re-sequencing is inherently more suitable for unstructured BPs containing knock-out tasks; because knock-out tasks are followed by splits, a branch of which can lead to immediate termination. On the other hand, parallelizing blocks of tasks and choosing their implementations benefits more from structured processes in our cases. Finally, note that elements, such as BPMN data objects, need not appear in the BPMN diagrams.

4.1 Processes as Annotated DAGs

We map processes P to a DAG $P' = (V', E')$, where $V' \supseteq V$ and $E' = (v'_i, v'_j),\ v'_i, v'_j \in V'$ (E' is neither a subset nor a superset of E). P' is annotated and more specifically, in our techniques in the next subsections, we require two quantitative metadata types for each vertex in P' in addition to the precedence constraints:

- *Cost* (c_i) per input token defines the cost of a task $v'_i,\ v'_i \in V',\ i = 1 \ldots |V'|$ to process a token. The cost can be either in time or in actual monetary cost units.
- *Selectivity* (sel_i) defines the (average) ratio of the output to the input tokens of v'_i. In many cases, the selectivity is 1, but lower selectivity values denote knock-out processes and/or are used to define the fraction of token flow after non-parallel gateways.

For the mapping purposes, we introduce the term *dummy tasks* [19]. These tasks represent artificial tasks that are characterized with appropriate statistical metadata to represent the flow of a BP. Finally, note that the edges in E' denote token flow exclusively.

Figures 4 and 5 show the corresponding DAG representations of each of the scenarios in Figures 1 and 3, respectively following the methodology in [19]. In the figures, we use example metadata. For instance, in Figure 4, the three checks have selectivity values 0.6, 0.8 and 0.5, respectively, whereas only 70% of the application that pass through these checks are finally approved. For the *Reimbursement Request* case study, the *Analyze Request* task has selectivity > 1, to account for the fact that a request may contain more than one separate claims.

4.2 Optimizations objectives

Cost-based optimization heavily relies on amortized metrics across several process instantiations. It is also helpful to define the *input (inp_i)* of a task. More

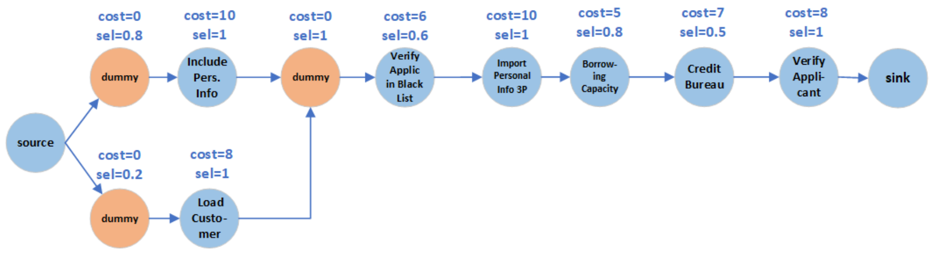


Fig. 4. A DAG representation of the model in Figure 1.

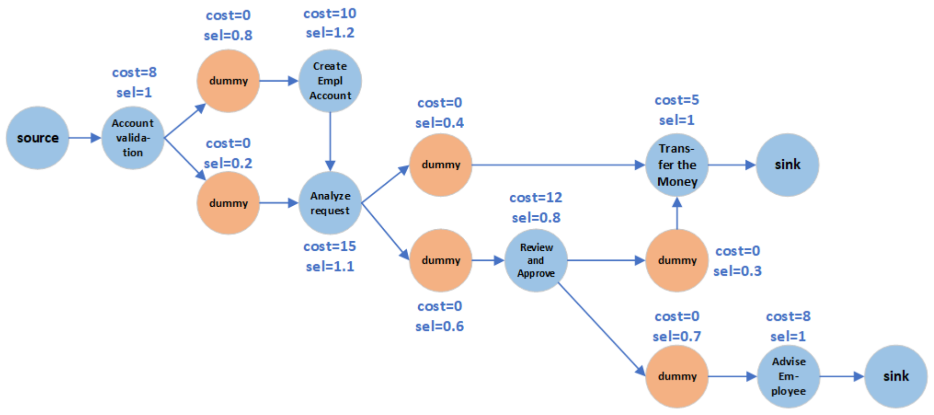


Fig. 5. A DAG representation of the model in Figure 3.

specifically, inp_i defines the size input of the v'_i in number of tokens. If v'_i receives input only from v'_j, then $inp_i = inp_j * sel_j$. In the generic case, $inp_i = \sum_{(v'_j, v'_i) \in E'} inp_j * sel_j$. In any case, inp_i depends on the product of the selectivities of the preceding tasks in the same path/branch in P'.

We consider the following two optimization objectives, which are common in BPs [17] (Ch. 7):

– *Monetary Cost/Resource Consumption* is defined by the human and machine costs. The human cost reflects the human resource consumption required to complete a BP execution, while the machine cost reflects the other resources that are necessary for the BP execution. This type of cost, given a set of tasks V' with known cost and selectivity values, is defined as follows:

$$Cost = \sum_{i=1}^{|V'|} c_i * inp_i$$

When c_i is in time units, the formula above computes the total time resources are occupied. If the cost is in monetary cost units, the formula computes the expenses associated with such resource usage.

– *Cycle time* represents the average time between the start of a process execution and its completion time. Generally, we define the cycle time as the sum of the costs of the tasks that belong to the critical path of a BP. The critical path is a path from a source to an end that takes the longest to complete. Taking the sum of the costs covers a wide range of cases, e.g., a model in the form of a chain, where each task is activated upon the receipt of a token, i.e., there are no waiting times or the (amortized) waiting times are included in task costs. If all tasks are executed by the same resource, then we need to sum the costs of all tasks, not only those in the critical path, to reflect the fact that tasks cannot be executed simultaneously.

5 Optimization Solutions

In this section, we provide algorithmic details and evaluation results regarding the two main problems we target in this work, namely reordering tasks in process models that contain knock-out tasks and treating bi-objective solutions regarding resource allocation in a principled manner.

5.1 Reordering knock-out tasks

First, we deal with reordering DAGs corresponding to BPs that contain knock-out tasks, i.e., tasks with selectivity not always set to 1 capitalizing on the state-of-art in data-centric workflow optimization, as the latter is presented in [42, 29] . Given the existence of efficient techniques for optimizing chains of tasks in data-centric workflows, the idea is to extract source-to-end linear segments (i.e., paths) corresponding to branches with knock-out tasks and then, optimize the extracted paths using existing techniques. In other words, given also that the problem is NP-hard [29], we follow a divide-and-conquer approach dealing with smaller parts of the DAG in each step to strike an acceptable balance between optimization overhead and solution optimality.

The important issue is that other types of tasks, i.e., non-selective ones, can exist as well. Let the *rank* value of a task be $\frac{1-sel_i}{c_i}$. It is well-known that ranking knock-out tasks according to their *rank* values yields an optimal solution in terms of the sum of the costs [5], but such an ordering is not always feasible due to precedence constraints. Therefore, more complete algorithms that build upon this principle have been developed with the proposals in [26] claiming to be the most advanced solutions to date. In addition, although the whole ordering can be done in several manners in polynomial time, for isolated paths with not a high number of tasks, exhaustive solutions are applicable in practice due to the efficient handling of the precedence constraints, as thoroughly discussed in [27]. A strong point of this approach is that the final ordering relies on the statistical metadata and precedence constraints rather than any initial designer decisions. More specifically, for a given linear segment, the initial ordering plays no role.

Algorithm 1 ReorderingBPs

```
1: Extract the largest source-to-end linear segment (path) from the DAG containing
   tasks with selectivity lower than 1.
2: if path size is shorter than (15 tasks) OR (shorter than 30 and DoF $< 0.1$) then
3:    execute Algorithm 2
4: else
5:    execute Algorithm 3
6: end if
7: remove all tasks in the selected path from the DAG and if there still exist tasks
   with selectivity <1 go to Step 1
```

Algorithm 2 TopSort (exact algorithm taken from [27])

```
Require: 1. A set of n tasks, T={$t_1$, ..., $t_n$}.
         2. A directed acyclic graph PC of precedence constraints.
         3. A function computeCost (the one in Section 4.2 can be used)
Ensure: An ordering of the tasks P representing the optimal plan.
1: P={$t_1$, $t_2$, ..., $t_n$} {P is initialized with a valid topological ordering of $PC$.}
2: i←1
3: minCost ← computeCost($P$)
4: while $i < n$ {n is the total number of tasks} do
5:    k ← location of $t_i$ in P
6:    k1 ← k + 1
7:    if $P(k1)$ task has prerequisite $t_i$ then
8:       // Rotation stage
9:       Rotate the elements of P from positions $i$ to $k$
10:      cost ← computeCost($P$)
11:      i← i+1
12:   else
13:      // Swapping stage
14:      Swap the $k$ and $k1$ elements of P
15:      cost ← computeCost($P$)
16:      i ← 1
17:   end if
18:   if cost < minCost then
19:      bestP ← P
20:      minCost ← cost
21:   end if
22: end while
23: P ← bestP
```

Solution Details The solution is outlined in Algorithm 1. This algorithm is a hybrid one and, as shown in line 2, defines the specific algorithm to be applied based on the properties of the path extracted. If the number of tasks in the extracted path is less than 15 or it is less than thirty with low task re-ordering flexibility, then the exact algorithm in Algorithm 2 is applied; otherwise we resort to a fast yet approximate solution (Algorithm 3). Later, we present experiments to explain the choice of these parameters. The flexibility in the ordering of the

Algorithm 3 RO-III (approximate algorithm taken from [26])

```
Require: A set of n tasks, T={$t_1$, ..., $t_n$}
    A directed acyclic graph PC of precedence constraints
Ensure: A directed acyclic graph P representing the optimal plan
 1: Find paths in PC sharing source and sink vertices
 2: Extract the innermost and most upstream set of paths with common sink and
    sources
 3: $PC \leftarrow$ merge all such paths in a common path according to their rank value
 4: Go to Step 1 until there are no other such paths in the modified PC
 5: $G \leftarrow$ Apply KBZ algorithm [31] given the modified precedence constraints
 6: repeat
 7:   {$k$ is the maximum subplan size considered}
 8:   for i←1:k do
 9:     for s←1:n-i do
10:       for t←s+i:n do
11:         consider moving subplan of size $i$ starting from the $s^{th}$ task after the $t^{th}$
            task in G
12:       end for
13:     end for
14:   end for
15: until no changes applied
16: P ← G
```

tasks is quantified with the help of the *DoF* (Degree-of-Freedom) metric. For a path with n tasks, when there is only a single alternative for task ordering, the corresponding transitive closure of the graph representing the precedence constraints has $n_{fullPCedges} = \frac{n(n-1)}{2}$ edges. Let $n_{PCedges}$ be the edges in the transitive closure of the precedence constraint graph. Then, $DoF = 1 - \frac{n_{PCedges}}{n_{fullPCedges}}$. Values closer to 1 denote maximum flexibility, e.g., as in the ad-hoc BPMN subprocesses, while values closer to 0 denote existence of few ordering alternatives.

Algorithm 2 presents the *TopSort* algorithm from [27] and is based on the solution from [51], which finds all the possible topological sortings given a partial ordering of a finite set. The algorithm is, by the problem definition, exponential in the worst case due to the size of the output (i.e., to enumerate all orderings), but, as will be shown, it is particularly useful in cases where the longest path of tasks does not comprise many tasks or there are highly restrictive precedence constraints. For the remaining cases, the authors advocate using Algorithm 3 from [26], termed as *ROIII.* This algorithm first renders the DAG applicable to a well-known optimization solution [31, 21] for database queries since the 80s, namely the so-called KBZ algorithm that was initially proposed for join ordering. This is done through manipulating the precedence constraints graph (lines 1-4) but comes at the expense of missing optimization opportunities. To ameliorate the latter drawback, the algorithm performs a heuristic post-processing step (lines 6-16). Overall, its complexity is $O(n^3)$ and for small k values in Algorithm 3, it becomes quadratic [26].

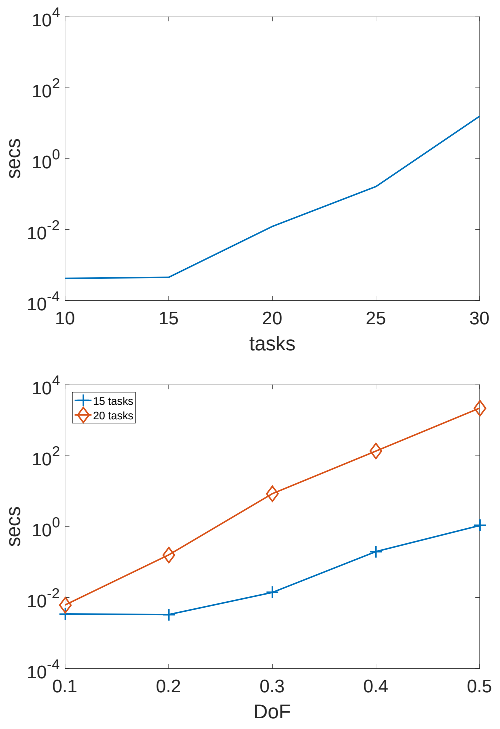


Fig. 6. TopSort scalability for varying number of tasks and DoF set to 0.1 (top) and varying DoF for 15 and 20 tasks (bottom).

To explain the switch condition in line 2 of Algorithm 1, the scalability of the exact and approximate algorithms need to be examined. In the experiments, a i7-4770 CPU at 3.4GHz with 16GB of RAM was used. As shown in Figure 6, the optimization overhead of *TopSort* grows significantly when the number of tasks exceeds 15, unless the *DoF* is kept low (< 0.1) where the tasks can be up to 30 and the optimization overhead is kept under 1 minute. For *TopSort*, the *DoF* value is important because it defines how many plans need to be checked. On the other hand, *ROIII* is more insensitive to the *DoF* value and even for 100 tasks, it runs in under a second (see Figure 7).

Example Benefits In order to assess the potential benefits of such re-ordering, we run the following experiment. The purpose of this experiment is to show that even in more conservative cases, the benefits are significant. More specifically, we consider two simple processes with just two and three knock-out tasks, respectively. For each task, the cost and the selectivity varies uniformly in the range [0.1, 1], which means that the costlier (resp. most selective) case differs from the less expensive (resp. less selective) by an order of magnitude; there are also start and end tasks with cost equal to the lowest of the knock-out ones (i.e., 0.1). The

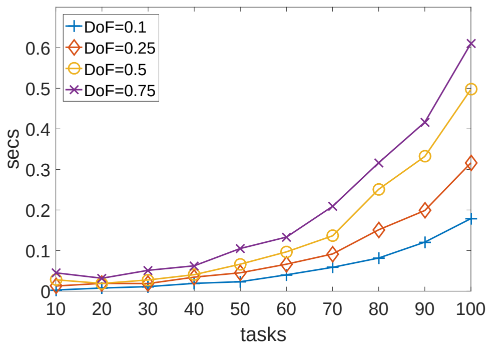


Fig. 7. ROIII scalability for varying number of tasks and DoF values.

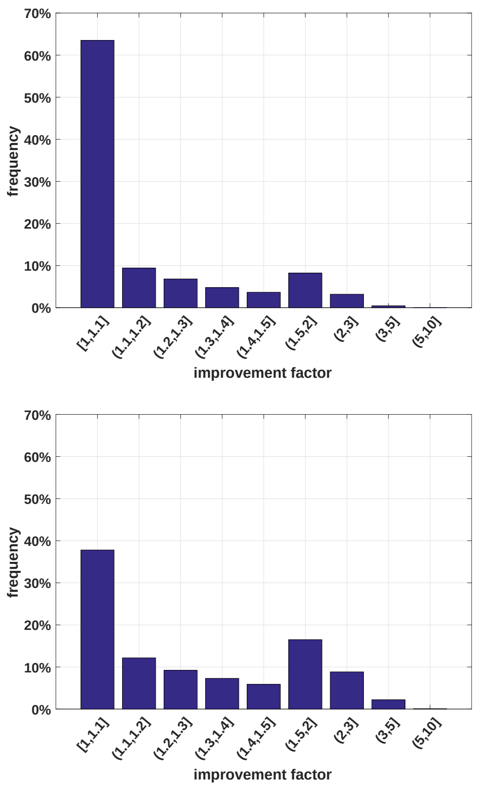


Fig. 8. Histograms regarding how many times the unoptimized process is slower than the optimized one when the process comprises solely two (top) and three (bottom) knock-out tasks.

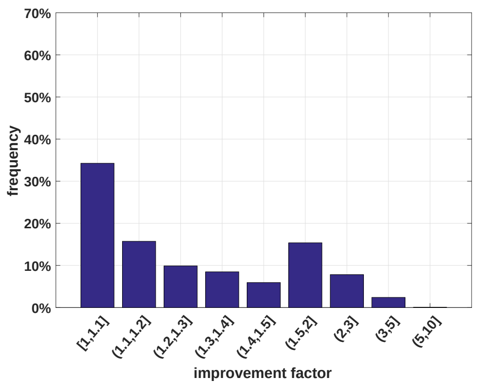


Fig. 9. Histogram regarding how many times the unoptimized process is slower than the optimized one when the process comprises four knock-out tasks with precedence constraints.

results are shown in Figure 8, where for each range of improvement values, we can see the probability of occurrence. When there are only two knock-out tasks, in 12% of the cases, the unoptimized plan has higher cost and cycle time by 50% and more (up to 3.9 times), as denoted by the cumulative frequency of the four rightmost bars in Figure 8(top). When there are three knock-out tasks, the non-optimized processes are inferior by more than 50% in 27.5% of the cases; the highest improvement observed is 6.79 times; see the four rightmost bars in Figure 8(bottom).

Next, we slightly modify the setting above as follows: we increase the number of tasks to four, but between 2 random pairs we enforce precedence constraints. The resulting histogram is shown in Figure 9. In this scenario, in 25% of the cases, the unnecessary overhead is higher than 50%. In the worst case, it is 5.68 times higher.

The benefits over a random initial ordering can grow arbitrarily large, if we consider more knock-out tasks within a process, with larger differences between the extremum cost and selectivity values.

Application to the Loan Application We apply the proposed technique to the load application scenario, where there are three knock-out tasks along with several other ones. We allow the costs and the selectivities of the tasks to differ by an order of magnitude at most (ranges [1, 10] and [0.1, 1], respectively) and we run 10000 random process instances. Figure 10 shows the speed-ups when the non-knock-out tasks have the minimum and the maximum costs. In the former case, the optimized plan is less expensive by up to 3.86 times and the benefits exceed 20% in more than 40% of the cases.

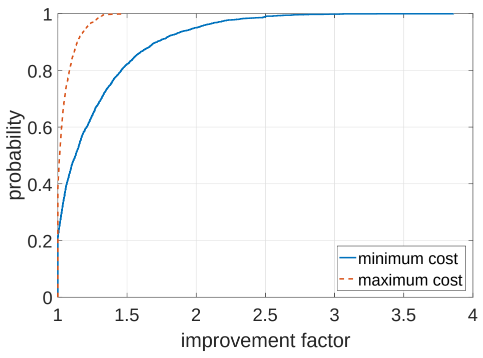


Fig. 10. The cumulative distribution function of the times of improvement for the loan application use case.

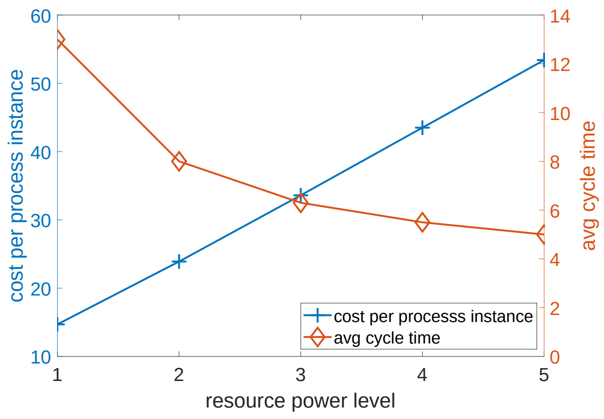


Fig. 11. Cost vs cycle-time trade-off example.

5.2 Dealing with optimization trade-offs

In the previous task re-ordering example, we essentially manipulate the set E of the BP edges. Next, we shift our attention to the properties of actors A, where we assume that the mapping of tasks to actors is already given through a function f, s.t. $f(V) \rightarrow A$. We are motivated by the fact that cycle time and cost are often contradictory objectives. For example, consider the scenario in Figure 3, where we assume that all tasks have cost equal to 1 cost unit and take 1 time unit to complete, apart from the review task that is 10 times costlier and more time consuming, and is allocated to a different actor. Further assume that we monitor the average performance across 100 input tokens (with 10 time units time gap between two instances). All requests are above $200 and approved.

Figure 11 shows the trade-off between cost and cycle-time when we increase the power of the bottleneck actor 2,3,4 and 5 times. When we double the power of the resource, the cost doubles and the time spent drops to the half. The results are derived using the BIMP simulator [2]. From the figure, we observe that the two cost metrics are monotonically changing and cycle time reaches a saturation point, so that allocating more powerful resources is meaningless. The exact balance between cost and cycle time to be decided depends on the weight of each optimization criterion. The above example is typical to several cases, but there are also several cases where the trade-offs are more complex and the solution space has less intuitive local and global optima, e.g., when the two cost metrics are combined in a non-linear objective function.

Allocating resources to a graph of connected tasks, so that the allocation regarding a specific task impacts on the other tasks, is a problem that has appeared in several flavors in data analytics, e.g., [36, 20, 30, 35], all of them corresponding to NP-hard problems. This implies that formulating the problem in a form, such as ILP, does not scale. Therefore, there are four main options remaining, using the work in [36] as an example of investigating all of them:

1. Heuristics that operate directly on the formal problem formulation, e.g., through restricting the possible allocation solutions to attain scalability.
2. Greedy heuristics.
3. Heuristics that are based on local search methods.
4. Other heuristics, mostly, nature-inspired ones [12].

The works in [36, 20, 30, 35], although they deal with different problem flavors, agree in that there is no *one-size-fits-all*, but in most cases the two main observations are: (i) greedy heuristics are very fast but they miss good trade-off between conflicting objectives; and (ii) local search methods are the most effective in striking good balance between such objectives. More importantly, local search methods are easy to develop and effective, provided that they start exploring the search point quite close to an acceptable final solution.

In summary, a high-level approach to handling the generic BP case can be as follows. We first compute the minimum cost and we define, through a user-defined threshold ϵ, the degradation in the cost that the BP designer can tolerate. Given the cost constraint, we search for the best cycle time. The exact optimality search policy is left for the user: in principle all brute force, greedy and more stochastic solutions apply. In cases where the initial solution, e.g., derived by rules or a polynomial algorithm or through optimizing one objective as above, is adequately satisfactory and it is reasonable to assume that a better solution can be produced through fine-tuning, methodologies that focus on the near neighborhood, such as iterated local search, are preferable. In cases where a vast search space should be covered efficiently, we should also employ methodologies, such as genetic algorithms.

6 Discussion

In this section, we discuss software development and open issues.

6.1 Development issues

For understanding the trade-offs when modifying the actor properties as discussed in Section 5.2, existing tools, such as the BIMP simulator [2], are adequate. However, automated re-ordering of tasks is a functionality that does not exist in modern tools for free. As a side-product of this manuscript, we have developed a software prototype that extends the Camunda BP management software [4] but can also support other BPM tools.

The software prototype consists of two main parts. The first part is a BPMN parser, which extracts the vertices that represent the tasks of the process, and the edges from the input BPMN file. To this end, we process the XML representation of the process model, which has the following form:

```
...
<bpmn:startEvent id="StartEvent_1">
  <bpmn:outgoing>SequenceFlow_11y212g</bpmn:outgoing>
</bpmn:startEvent>
<bpmn:task id="A" name="Task A">
  <bpmn:incoming>SequenceFlow_11y212g</bpmn:incoming>
  <bpmn:outgoing>SequenceFlow_02dhphd</bpmn:outgoing>
</bpmn:task>
...
<bpmn:endEvent id="EndEvent_1m1wp9j">
  <bpmn:incoming>SequenceFlow_1cm715l</bpmn:incoming>
</bpmn:endEvent>
<bpmn:sequenceFlow id="SequenceFlow_11y212g"
 sourceRef="StartEvent_1" targetRef="A" />
...
```

In the above example, a start event is followed by a task with id A. Essentially, the Camunda parser searches for `bpmn:startEvent`, `bpmn:endEvent`, `bpmn:task` and `bpmn:sequenceFlow` XML elements.

The second part takes as input the output of the first part and additionally the metadata regarding the corresponding BP, namely, task selectivity, cost of execution and precedence constraints, which are assumed to be known. Finally, the process is optimized according to the algorithm presented and is output as a BPMN XML file, where the `bpmn:sequenceFlow`, `bpmn:incoming` and `bpmn:outgoing` elements are defined according to the optimized process model.

The prototype mentioned above is capable of performing automated optimization, provided that the required metadata exist. Following up on the discussion in Section 2, to date, there is not a clear solution to the issue of their automated extraction. In order to solve this issue, we advocate an approach that relies on BP event log analysis and/or declarative process modeling, as in [38, 37, 33].

A set of BP event log analysis techniques that aim to extract BP metadata have been proposed. For example, in [6], the replay method is presented, where, after the process model extraction from the logs is complete, every event log case is attempted to be "replayed" (i.e. re-executed) using each of the extracted models. As a result, the fitness of each of the extracted models is calculated. Additionally, this method can be used to collect various type of statistics regarding the execution of the process. These include task duration, which is related to execution cost, and routing probabilities (e.g. in OR gateways) which is related to selectivity. Regarding precedence constraints, the focus is shifted towards behavioral relations that exist between tasks [48, 41]. In both approaches, a set of relations (i.e. patterns) are extracted from the event logs; these patterns contain the precedence constraints that exist between the tasks of the process. The complete investigation of the issue of automated metadata extraction capitalizing on existing process mining solutions is complementary to this work and is left as a future extension. Finally, the systematic manual specification of the constraints has already been proposed in [38, 37]. The drawback is that the burden is shifted to the process designer to specify such constraints.

6.2 Open issues

In the previous sections, we provided concrete examples of application of optimization techniques initially proposed for data-centric workflows to BPs along with insights into performance benefits. However, there are several open issues in providing end-to-end solutions. We elaborate on these issues below:

- Even the better investigated type of data-centric workflow optimization, namely task re-ordering, requires extensions to cover generic BP scenarios. For example, in the model in Figure 3, an optimization might move the expensive supervisor check task in parallel with the check for account existence to decrease cycle time. The current state-of-the-art in algorithms for minimizing response time in data analytic workflows do not account for such model structure modifications in an automated manner [28]; therefore further research is required in this direction.
- In data-centric dataflows, the equivalent of a token is a data record and, typically, we are interested in the performance when processing a large number of such records. The main difference to BPs is that all input records are available at the beginning of the workflow execution; by contrast, in BPs, the tokens arrive according to several types of probability distributions. In addition, in data-centric workflows, it is important to measure the time of processing for the whole input dataset, whereas in BPs, the cycle time refers to the time of processing a single input token taking into consideration any waiting times. The cost models in data-centric workflows [29] need to be extended to better cover these aspects, e.g., nowadays they are inadequate to cover all metrics in the BIMP simulator [2].
- In the optimization algorithms presented in Section 5.1, we split BP graphs into branches. For developing more holistic solutions, we need further constraint types apart from precedence ones, e.g., to specify that two tasks cannot belong to the same branch. Then, algorithms that can handle such constraints need to be developed.

– The presented solutions rely on the existence of statistical metadata as well; thus extending the current techniques for process monitoring to support metadata extraction is required for rendering the proposed solutions practical as discussed in Section 6. Orthogonally, flexibility is a key objective in BPs (see Ch. 8 in [17]); in terms of optimization techniques, flexibility can be manifested in developing techniques that are robust to small changes in the metadata values. Developing solutions that optimize also for flexibility is an interesting direction for future work that can also benefit from advances in the data management community [53].
– The optimization solutions presented are automated but this does not imply that they can enact automated optimizations in the actual running business processes in an organization. For example, the algorithm may advocate doubling the capacity of a specific resource type. However, they do not specify whether this can be achieved by modifying the duties of personnel, hiring new staff, upgrading some software system and so on.
– The mapping process from BPMN to DAGs in Section 4.1 has not been explored thoroughly and validated in arbitrary scenarios. More work is needed in order not only to map BP models to DAGs but also to map the optimized DAG back to a BP model. This needs to be coupled with mechanisms for eligibility assessment of BP models as to whether it is possible and/or meaningful to perform the mapping (see [49] for an early porposal on these issues).
– This methodology can be easily adapted to other process modelling languages and approaches, such as refined process trees [50], where each block can be mapped to a vertex, and Event-driven Process Chains. In the future, we plan to investigate this issue in depth.

7 Conclusions

In this work, we aim to introduce the benefits from transferring big data technologies to business process optimization, in line with the broader vision in [44]. Process mining already plays a significant role in business process management; the key idea behind this work is that, apart from process mining, advanced data management solutions can offer many additional benefits when transferred to business process scenarios. We specifically focus on optimizing aspects such as total cost and cycle time, for which the counterpart techniques in data-centric workflows seem more mature. Data-centric workflows traditionally emphasize on improvements in large flows processing huge amounts of data; however, here we aim to show that they are useful even in small real-life processes as well, yielding improvements up to several times, especially when there are knock-out tasks with flexible ordering. We present complete algorithmic solutions for reordering tasks and systematically exploring the search space of possible solutions under two conflicting objectives, whereas we also describe software implementation aspects of our proposal. For devising end-to-end solutions, we discuss issues remaining open, which range from new algorithms to further work on modeling and process monitoring to collect the necessary metadata.

References

1. Appian: Low-code platform and bpm software for digital transformation. https://www.appian.com/
2. BIMP - the business process simulator. http://bimp.cs.ut.ee/
3. Bizagi - digital transformation and business process management bpm. https://www.bizagi.com/en
4. Camunda bpm: Workflow and decision automation platform. https://camunda.com/
5. van der Aalst, W.M.P.: Re-engineering knock-out processes. Decis. Support Syst. 30(4), 451–468 (2001)
6. van der Aalst, W.M.P.: Process Mining - Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
7. van der Aalst, W.M.P.: Spreadsheets for business process management: Using process mining to deal with "events" rather than "numbers"? Business Proc. Manag. Journal 24(1), 105–127 (2018)
8. Agrawal, K., Benoit, A., Dufossé, F., Robert, Y.: Mapping filtering streaming applications. Algorithmica 62(1-2), 258–308 (2012)
9. Augusto, A., Conforti, R., Dumas, M., Rosa, M.L.: Split miner: Discovering accurate and simple business process models from event logs. In: 2017 IEEE International Conference on Data Mining, ICDM 2017, New Orleans, LA, USA, November 18-21, 2017. pp. 1–10 (2017)
10. Augusto, A., Conforti, R., Dumas, M., Rosa, M.L., Bruno, G.: Automated discovery of structured process models from event logs: The discover-and-structure approach. Data Knowl. Eng. 117, 373–392 (2018)
11. Bose, R.P.J.C., van der Aalst, W.M.P.: Trace alignment in process mining: Opportunities for process diagnostics. In: Business Process Management - 8th International Conference, BPM 2010, Hoboken, NJ, USA, September 13-16, 2010. Proceedings. pp. 227–242 (2010)
12. Brownlee, J.: Clever algorithms: nature-inspired programming recipes. Jason Brownlee (2011)
13. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: Discovering and navigating a collection of process models using multiple quality dimensions. In: Business Process Management Workshops - BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers. pp. 3–14 (2013)
14. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: Mining configurable process models from collections of event logs. In: Business Process Management - 11th International Conference, BPM 2013, Beijing, China, August 26-30, 2013. Proceedings. pp. 33–48 (2013)
15. Buzacott, J.A.: Commonalities in reengineered business processes: Models and issues. Management Science 42(5), 768–782 (1996)
16. Deshpande, A., Hellerstein, L.: Parallel pipelined filter ordering with precedence constraints. ACM Trans. Algorithms 8(4), 41:1–41:38 (2012)
17. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management, Second Edition. Springer (2018)
18. Falk, T., Griesberger, P., Leist, S.: Patterns as an artifact for business process improvement-insights from a case study. In: International Conference on Design Science Research in Information Systems. pp. 88–104. Springer (2013)
19. Gounaris, A.: Towards automated performance optimization of BPMN business processes. In: New Trends in Databases and Information Systems - ADBIS 2016 Short Papers and Workshops. pp. 19–28 (2016)

20. Gounaris, A., Kougka, G., Tous, R., Montes, C.T., Torres, J.: Dynamic configuration of partitioning in spark applications. IEEE Trans. Parallel Distrib. Syst. 28(7), 1891–1904 (2017)
21. Ibaraki, T., Kameda, T.: On the optimal nesting order for computing n-relational joins. ACM Trans. Database Syst. 9(3), 482–502 (1984)
22. Indulska, M., zur Muehlen, M., Recker, J.: Measuring method complexity: The case of the business process modeling notation. Tech. Rep. BPM Center Report BPM-09-03, , BPMcenter.org (2009)
23. Jennings, N.R., Norman, T.J., Faratin, P., O'Brien, P., Odgers, B.: Autonomous agents for business process management. Applied Artificial Intelligence 14(2), 145–189 (2000)
24. Kiepuszewski, B., Hofstede, A.H.M.t., Bussler, C.: On structured workflow modelling. In: Proceedings of the 12th International Conference on Advanced Information Systems Engineering. pp. 431–445. CAiSE '00 (2000)
25. Köpke, J., Franceschetti, M., Eder, J.: Optimizing data-flow implementations for inter-organizational processes. Distributed and Parallel Databases (2018)
26. Kougka, G., Gounaris, A.: Cost optimization of data flows based on task re-ordering. T. Large-Scale Data- and Knowledge-Centered Systems 33, 113–145 (2017)
27. Kougka, G., Gounaris, A.: Optimal task ordering in chain data flows: Exploring the practicality of non-scalable solutions. In: Big Data Analytics and Knowledge Discovery - 19th International Conference, DaWaK 2017, Lyon, France, August 28-31, 2017, Proceedings. vol. 10440, pp. 19–32 (2017)
28. Kougka, G., Gounaris, A.: Optimization of data flow execution in a parallel environment. Distributed and Parallel Databases (2018), https://doi.org/10.1007/s10619-018-7243-3
29. Kougka, G., Gounaris, A., Simitsis, A.: The many faces of data-centric workflow optimization: a survey. I. J. Data Science and Analytics 6(2), 81–107 (2018)
30. Kougka, G., Gounaris, A., Tsichlas, K.: Practical algorithms for execution engine selection in data flows. Future Generation Comp. Syst. 45, 133–148 (2015)
31. Krishnamurthy, R., Boral, H., Zaniolo, C.: Optimization of nonrecursive queries. In: VLDB. pp. 128–137 (1986)
32. La Rosa, M., Dumas, M., ter Hofstede, A.H.M., Mendling, J.: Configurable multi-perspective business process models. Inf. Syst. 36(2), 313–340 (2011)
33. Maggi, F.M., Mooij, A.J., van der Aalst, W.M.P.: User-guided discovery of declarative process models. In: Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2011, part of the IEEE Symposium Series on Computational Intelligence 2011, April 11-15, 2011, Paris, France. pp. 192–199 (2011)
34. Mendling, J.: Metrics for Process Models: Empirical Foundations of Verification, Error Prediction, and Guidelines for Correctness, Lecture Notes in Business Information Processing, vol. 6. Springer (2008)
35. Michailidou, A., Gounaris, A.: Bi-objective traffic optimization in geo-distributed data flows. Big Data Res. 16, 36–48 (2019)
36. Nardelli, M., Cardellini, V., Grassi, V., Presti, F.L.: Efficient operator placement for distributed data stream processing applications. IEEE Trans. Parallel Distrib. Syst. 30(8), 1753–1767 (2019)
37. Pesic, M., van der Aalst, W.M.P.: A declarative approach for flexible business processes management. In: Business Process Management Workshops, BPM. pp. 169–180 (2006)

38. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: full support for loosely-structured processes. In: 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007). pp. 287–300 (2007)
39. Polyvyanyy, A., García-Bañuelos, L., Dumas, M.: Structuring acyclic process models. Inf. Syst. 37(6), 518–538 (2012)
40. Polyvyanyy, A., Ouyang, C., Barros, A., van der Aalst, W.M.P.: Process querying: Enabling business intelligence through query-based process analytics. Decision Support Systems 100, 41–56 (2017)
41. Pourmasoumi, A., Bagheri, E.: Business process mining. CoRR abs/1607.00607 (2016)
42. Rheinländer, A., Leser, U., Graefe, G.: Optimization of complex dataflows with user-defined functions. ACM Comput. Surv. 50(3), 38:1–38:39 (2017)
43. Rosa, M.L., Wohed, P., Mendling, J., ter Hofstede, A.H.M., Reijers, H.A., van der Aalst, W.M.P.: Managing process model complexity via abstract syntax modifications. IEEE Trans. Industrial Informatics 7(4), 614–629 (2011)
44. Sakr, S., Maamar, Z., Awad, A., Benatallah, B., van der Aalst, W.M.P.: Business process analytics and big data systems: A roadmap to bridge the gap. IEEE Access 6, 77308–77320 (2018)
45. Schunselaar, D.: Configurable process trees: elicitation, analysis, and enactment (2016)
46. Simitsis, A., Wilkinson, K., Castellanos, M., Dayal, U.: Optimizing analytic data flows for multiple execution engines. In: SIGMOD Conference. pp. 829–840 (2012)
47. Simitsis, A., Wilkinson, K., Dayal, U., Castellanos, M.: Optimizing ETL workflows for fault-tolerance. In: ICDE. pp. 385–396 (2010)
48. Tao, J., Deokar, A.V.: An organizational mining approach based on behavioral process patterns. In: 20th Americas Conference on Information Systems, AMCIS 2014, Savannah, Georgia, USA, August 7-9, 2014 (2014)
49. Tsakalidis, G., Vergidis, K., Kougka, G., Gounaris, A.: Eligibility of BPMN models for business process redesign. Information 10(7), 225 (2019)
50. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. In: Business Process Management, 6th International Conference, BPM 2008, Milan, Italy, September 2-4, 2008. Proceedings. pp. 100–115 (2008)
51. Varol, Y.L., Rotem, D.: An algorithm to generate all topological sorting arrangements. The Computer Journal 24(1), 83–84 (1981)
52. Vergidis, K., Tiwari, A., Majeed, B.: Business process analysis and optimization: Beyond reengineering. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 38(1), 69–82 (2008)
53. Wolf, F., Brendle, M., May, N., Willems, P.R., Sattler, K., Grossniklaus, M.: Robustness metrics for relational query execution plans. PVLDB 11(11), 1360–1372 (2018)
54. Yilmaz, O., Karagoz, P.: Generating performance improvement suggestions by using cross-organizational process mining. In: Proceedings of the 5th International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2015), Vienna, Austria, December 9-11, 2015. pp. 3–17 (2015)
55. Zaharia, M., Xin, R.S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M.J., Ghodsi, A., Gonzalez, J., Shenker, S., Stoica, I.: Apache spark: a unified engine for big data processing. Commun. ACM 59(11), 56–65 (2016)
56. van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.P.: Event stream-based process discovery using abstract representations. Knowl. Inf. Syst. 54(2), 407–435 (2018)